

Online Learning for Wireless Distributed Computing

Yi-Hsuan Kao*, Kwame Wright*, Bhaskar Krishnamachari* and Fan Bai†

*Ming-Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA, USA
Email: {yihsuank, kwamelaw, bkrishna}@usc.edu

†General Motors Global R&D
Warren, MI, USA
Email: fan.bai@gm.com

Abstract—There has been a growing interest for Wireless Distributed Computing (WDC), which leverages collaborative computing over multiple wireless devices. WDC enables complex applications that a single device cannot support individually. However, the problem of assigning tasks over multiple devices becomes challenging in the dynamic environments encountered in real-world settings, considering that the resource availability and channel conditions change over time in unpredictable ways due to mobility and other factors. In this paper, we formulate a task assignment problem as an online learning problem using an adversarial multi-armed bandit framework. We propose MABSTA, a novel online learning algorithm that learns the performance of unknown devices and channel qualities continually through exploratory probing and makes task assignment decisions by exploiting the gained knowledge. For maximal adaptability, MABSTA is designed to make no stochastic assumption about the environment. We analyze it mathematically and provide a worst-case performance guarantee for any dynamic environment. We also compare it with the optimal offline policy as well as other baselines via emulations on trace-data obtained from a wireless IoT testbed, and show that it offers competitive and robust performance in all cases. To the best of our knowledge, MABSTA is the first online algorithm in this domain of task assignment problems and provides provable performance guarantee.

I. INTRODUCTION

We are at the cusp of revolution as the number of connected devices is projected to grow significantly in the near future. These devices, either suffering from stringent battery usage, like mobile devices, or limited processing power, like sensors, are not capable to run computation-intensive tasks independently. Nevertheless, what can these devices do if they are connected and collaborate with each other? The connected devices in the network, sharing resources with each other, provide a platform with abundant computational resources that enables the execution of complex applications [1], [2].

Traditional cloud services provide access to high performance and reliable servers. However, considering the varying link quality and the long run trip times (RTTs) of a wide-area network (WAN) and possibly long setup time, these remote servers might not always be the best candidates to help in scenarios where the access delay is significant [3], [4]. Another approach is to exploit nearby computational resources, including mobile devices, road-side units (RSUs) and local servers. These devices are not as powerful as cloud servers in general, but can be accessed by faster device to device (D2D) communication [5]. In addition to communication over varying wireless links, the workload on a device also affects the amount of resource it can release. Hence, a system has to identify the

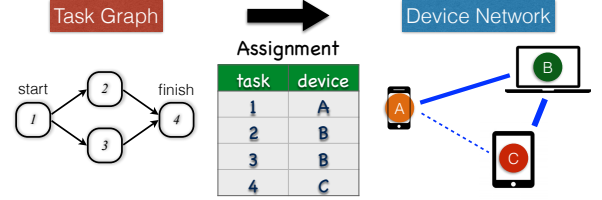


Fig. 1: An application consists of multiple tasks. In order to perform collaborative computing over heterogeneous devices connected in the network, a system has to find out a good task assignment strategy, considering devices' feature, workload and different channel qualities between them.

available resources in the network and decide how to leverage them among a number of possibilities, considering the dynamic environment at run time.

Figure 1 illustrates the idea of Wireless Distributed Computing. Given an application that consists of multiple tasks, we want to assign them on multiple devices, considering the resource availability so that the system performance, in metrics like energy consumption and application latency, can be improved. These resources that are accessible by wireless connections form a resource network, which is subject to frequent topology changes and has the following features:

Dynamic device behavior: The quantity of the released resource varies with devices, and may also depend on the local processes that are running. Moreover, some of devices may carry microprocessors that are specialized in performing a subset of tasks. Hence, the performance of each device varies highly over time and different tasks and is hard to model as a known and stationary stochastic process.

Heterogeneous network with intermittent connections: Devices' mobility makes the connections intermittent, which change drastically in quality within a short time period. Furthermore, different devices may use different protocols to communicate with each other. Hence, the performance of the links between devices is also highly dynamic and variable and hard to model as a stationary process.

A. Why online learning?

From what we discuss above, since the resource network is subject to drastic changes over time and is hard to be modeled by stationary stochastic processes, we need an algorithm that

applies to *all* possible scenarios, learns the environment at run time, and adapts to changes. Existing works focus on solving optimization problems given known deterministic profile or known stochastic distributions [6], [7]. These problems are hard to solve. More importantly, algorithms that lack learning ability could be harmed badly by statistical changes or mismatch between the profile (offline training) and the run-time environment. Hence, we use an online learning approach, which takes into account the performance during the learning phase, and aim to learn the environment quickly and adapt to changes.

We formulate the task assignment problem as an adversarial multi-armed bandit (MAB) problem that does not make any stochastic assumptions on the resource network [8]. We propose MABSTA (Multi-Armed Bandit based Systematic Task Assignment) that learns the environment and makes task assignment at run time. Furthermore, We provide worst-case analysis on the performance to guarantee that MABSTA performs no worse than a provable lower bound in *any* dynamic environment. To the best of our knowledge, MABSTA is the first online algorithm in this domain of task assignment problems and provides provable performance guarantee.

B. Contributions

A new formulation of task assignment problems considering general and dynamic environment: We use a novel adversarial multi-armed bandit (MAB) formulation that does not make any assumptions on the dynamic environment. That is, it applies to all realistic scenarios.

A light algorithm that learns the environment quickly with provable performance guarantee: MABSTA runs with light complexity and storage, and admits performance guarantee and learning time that are significantly improved compared to the existing MAB algorithm.

Broad applications on wireless device networks: MABSTA enhances collaborative computing over wireless devices, enabling more potential applications on mobile cloud computing, wireless sensor networks and Internet of Things.

II. BACKGROUND ON MULTI-ARMED BANDIT PROBLEMS

The multi-armed bandit (MAB) problem is a sequential decision problem where at each time an agent chooses over a set of “arms”, gets the payoff from the selected arms and tries to learn the statistical information from sensing them. These formulations have been considered recently in the context of opportunistic spectrum access for cognitive radio wireless networks, but those formulations are quite different from ours in that they focus only on channel allocation and not on also allocating computational tasks to servers [9], [10].

Given an online algorithm to a MAB problem, its performance is measured by a regret function, which specifies how much the agent loses due to the unknown information at the beginning [11]. For example, we can compare the performance to a genie who knows the statistics of payoff functions and selects the arms based on the best policy.

Stochastic MAB problems model the payoff of each arm as a stationary random process and aim to learn the unknown information behind it. If the distribution is unknown but is

known to be i.i.d. over time, Auer *et al.* [12] propose UCB algorithms to learn the unknown distribution with bounded regret. However, the assumption on i.i.d. processes does not always apply to the real environment. On the other hand, Ortner *et al.* [13] assume the distribution is known to be a Markov process and propose an algorithm to learn the unknown state transition probabilities. However, the large state space of Markov process causes our task assignment problem to be intractable. Hence, we need a *tractable* algorithm that applies to stochastic processes with *relaxed* assumptions on time-independence stationarity.

Adversarial MAB problems, however, do not make any assumptions on the payoffs. Instead, an agent learns from the sequence given by an adversary who has complete control over the payoffs [8]. In addition to the well-behaved stochastic processes, an algorithm of adversarial MAB problems gives a solution that generally applies to all bounded payoff sequences and provides the the worst-case performance guarantee.

Auer *et al.* [14] propose Exp3, which serves adversarial MAB and yields a sub-linear regret with time ($O(\sqrt{T})$). That is, compared to the optimal offline algorithm, Exp3 achieves asymptotically 1-competitive. However, if we apply Exp3 to our task assignment problem, there will be an exponential number of arms, hence, the regret will grow exponentially with problem size. In this paper, we propose an algorithm providing that the regret is not only bounded by $O(\sqrt{T})$ but also bounded by a polynomial function of problem size.

III. PROBLEM FORMULATION

Suppose a data processing application consists of N tasks, where their dependencies are described by a directed acyclic graph (DAG) $G = (\mathcal{V}, \mathcal{E})$ as shown in Figure 1. That is, an edge (m, n) implies that some data exchange is necessary between task m and task n and hence task n cannot start until task m finishes. There is an incoming data stream to be processed (T data frames in total), where for each data frame t , it is required to go through all the tasks and leave afterwards. There are M available devices. The assignment strategy of data frame t is denoted by a vector $\mathbf{x}^t = x_1^t, \dots, x_N^t$, where x_i^t denotes the device that executes task i . Given an assignment strategy, stage-wised costs apply to each node (task) for computation and each edge for communication. The cost can correspond to the resource consumption for a device to complete a task, for example, energy consumption.

In the following formulation we follow the tradition in MAB literature and focus on maximizing a positive reward instead of minimizing the total cost, but of course these are mathematically equivalent, e.g., by setting $\text{reward} = \text{maxCost} - \text{cost}$. When processing data frame t , let $R_i^{(j)}(t)$ be the reward of executing task i on device j . Let $R_{mn}^{(jk)}(t)$ be the reward of transmitting the data of edge (m, n) from device j to k . The reward sequences are unknown but are bounded between 0 and 1. Our goal is to find out the assignment strategy for each data frame based on the previously observed samples, and compare the performance with a genie that uses the best assignment strategy for all data frames. That is,

$$R_{total}^{max} = \max_{\mathbf{x} \in \mathcal{F}} \sum_{t=1}^T \left(\sum_{i=1}^N R_i^{(x_i)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(x_m x_n)}(t) \right), \quad (1)$$

Algorithm 1 MABSTA

```

1: procedure MABSTA( $\gamma, \alpha$ )
2:    $w_{\mathbf{y}}(1) \leftarrow 1 \forall \mathbf{y} \in \mathcal{F}$ 
3:   for  $t \leftarrow 1, 2, \dots, T$  do
4:      $W_t \leftarrow \sum_{\mathbf{y} \in \mathcal{F}} w_{\mathbf{y}}(t)$ 
5:     Draw  $\mathbf{x}^t$  from distribution
        
$$p_{\mathbf{y}}(t) = (1 - \gamma) \frac{w_{\mathbf{y}}(t)}{W_t} + \frac{\gamma}{|\mathcal{F}|} \quad (2)$$

6:     Get rewards  $\{R_i^{(x_i^t)}(t)\}_{i=1}^N, \{R_{mn}^{(x_m^t x_n^t)}(t)\}_{(m,n) \in \mathcal{E}}$ .
7:      $\mathcal{C}_{ex}^i \leftarrow \{\mathbf{z} \in \mathcal{F} | z_i = x_i^t, \forall i\}$ 
8:      $\mathcal{C}_{tx}^{mn} \leftarrow \{\mathbf{z} \in \mathcal{F} | z_m = x_m^t, z_n = x_n^t\}, \forall (m, n)$ 
9:     for  $\forall j \in [M], \forall i \in [N]$  do
        
$$\hat{R}_i^{(j)}(t) = \begin{cases} \frac{R_i^{(j)}(t)}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}(t)} & \text{if } x_i^t = j, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

10:    end for
11:    for  $\forall j, k \in [M], \forall (m, n) \in \mathcal{E}$  do
        
$$\hat{R}_{mn}^{(jk)}(t) = \begin{cases} \frac{R_{mn}^{(jk)}(t)}{\sum_{\mathbf{z} \in \mathcal{C}_{tx}^{mn}} p_{\mathbf{z}}(t)} & \text{if } x_m^t = j, x_n^t = k, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

12:    end for
13:    Update for all  $\mathbf{y}$ 
        
$$\hat{R}_{\mathbf{y}}(t) = \sum_{i=1}^N \hat{R}_i^{(y_i)}(t) + \sum_{(m,n) \in \mathcal{E}} \hat{R}_{mn}^{(y_m y_n)}(t), \quad (5)$$

        
$$w_{\mathbf{y}}(t+1) = w_{\mathbf{y}}(t) \exp(\alpha \hat{R}_{\mathbf{y}}(t)). \quad (6)$$

14:  end for
15: end procedure

```

where \mathcal{F} represents the set of feasible solutions. The genie who knows all the reward sequences can find out the best assignment strategy, however, not knowing these sequences in advance, our proposed online algorithm aims to learn this best strategy and remain competitive in overall performance.

IV. MABSTA ALGORITHM

We summarize MABSTA in Algorithm 1. For each data frame t , MABSTA randomly selects a feasible assignment (arm $\mathbf{x} \in \mathcal{F}$) from a probability distribution that depends on the weights of arms ($w_{\mathbf{y}}(t)$). Then it updates the weights based on the reward samples. From (2), MABSTA randomly switches between two phases: exploitation (with probability $1 - \gamma$) and exploration (with probability γ). At exploitation phase, MABSTA selects an arm based on its weight. Hence, the one with higher reward samples will be chosen more likely. At exploration phase, MABSTA uniformly selects an arm without considering its performance. The fact that MABSTA keeps probing every arms makes it adaptive to the changes of the environment, compared to the case where static strategy plays the previously best arm all the time without knowing that other arms might have performed better currently.

The commonly used performance measure for an MAB algorithm is its regret. In our case it is defined as the difference in accumulated rewards (\hat{R}_{total}) compared to a genie that

knows all the rewards and selects a single best strategy for all data frames (R_{total}^{max} in (1)). Auer *et al.* [14] propose Exp3 for adversarial MAB. However, if we apply Exp3 to our online task assignment problem, since we have an exponential number of arms (M^N), the regret bound will grow exponentially. The following theorem shows that MABSTA guarantees a regret bound that is polynomial with problem size and $O(\sqrt{T})$.

Theorem 1. Assume all the reward sequences are bounded between 0 and 1. Let \hat{R}_{total} be the total reward achieved by Algorithm 1. For any $\gamma \in (0, 1)$, let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, we have

$$R_{total}^{max} - \mathbb{E}\{\hat{R}_{total}\} \leq (e-1)\gamma R_{total}^{max} + \frac{M(N+|\mathcal{E}|M) \ln M^N}{\gamma}.$$

In above, N is the number of nodes (tasks) and $|\mathcal{E}|$ is the number of edges in the task graph. We leave the proof of Theorem 1 in the appendix. By applying the appropriate value of γ and using the upper bound $R_{total}^{max} \leq (N+|\mathcal{E}|)T$, we have the following Corollary.

Corollary 1. Let $\gamma = \min\{1, \sqrt{\frac{M(N+|\mathcal{E}|M) \ln M^N}{(e-1)(N+|\mathcal{E}|)T}}\}$, then

$$R_{total}^{max} - \mathbb{E}\{\hat{R}_{total}\} \leq 2.63\sqrt{(N+|\mathcal{E}|)(N+|\mathcal{E}|M)MNT \ln M}.$$

We look at the worst case, where $|\mathcal{E}| = O(N^2)$. The regret can be bounded by $O(N^{2.5}MT^{0.5})$. Since the bound is a concave function of T , we define the learning time T_0 as the time when its slope falls below a constant c . That is,

$$T_0 = \frac{1.73}{c^2} (N+|\mathcal{E}|)(N+|\mathcal{E}|M)MN \ln M.$$

This learning time is significantly improved compared with applying Exp3 to our problem, where $T_0 = O(M^N)$. As we will show in the numerical results, MABSTA performs significantly better than Exp3 in the trace-data emulation.

V. POLYNOMIAL TIME MABSTA

In Algorithm 1, since there are exponentially many arms, implementation may result in exponential storage and complexity. However, in the following, we propose an equivalent but efficient implementation. We show that when the task graph belongs to a subset of DAG that appear in practical applications (namely, parallel chains of trees), Algorithm 1 can run in polynomial time with polynomial storage.

We observe that in (5), $R_{\mathbf{y}}(t)$ relies on the estimates of each node and each edge. Hence, we rewrite (6) as

$$\begin{aligned} w_{\mathbf{y}}(t+1) &= \exp\left(\alpha \sum_{\tau=1}^t R_{\mathbf{y}}(\tau)\right) \\ &= \exp\left(\alpha \sum_{i=1}^N \tilde{R}_i^{(y_i)}(t) + \alpha \sum_{(m,n) \in \mathcal{E}} \tilde{R}_{mn}^{(y_m y_n)}(t)\right), \end{aligned} \quad (7)$$

where

$$\tilde{R}_i^{(y_i)}(t) = \sum_{\tau=1}^t \hat{R}_i^{(y_i)}, \quad \tilde{R}_{mn}^{(y_m y_n)}(t) = \sum_{\tau=1}^t \hat{R}_{mn}^{(y_m y_n)}.$$

Algorithm 2 Calculate $w_N^{(j)}$ for tree-structured task graph

```

1: procedure  $\Omega(N, M, G)$ 
2:    $q \leftarrow \text{BFS}(G, N) \triangleright$  run BFS from  $N$  and store visited
   nodes in order
3:   for  $i \leftarrow q.\text{end}, q.\text{start}$  do  $\triangleright$  start from the last element
4:     if  $i$  is a leaf then  $\triangleright$  initialize  $\omega$  values of leaves
5:        $\omega_i^{(j)} \leftarrow e_i^{(j)}$ 

6:   else
7:      $\omega_i^{(j)} \leftarrow e_i^{(j)} \prod_{m \in \mathcal{N}_i} \sum_{y_m \in [M]} e_{mi}^{(y_m j)} \omega_m^{(y_m)}$ 

8:   end if
9: end for
10: end procedure
  
```

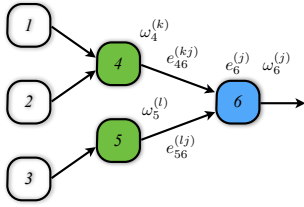


Fig. 2: An example of tree-structure task graph, where $\mathcal{D}_6 = \{1, 2, 3, 4, 5\}$, and $\mathcal{E}_6 = \{(1, 4), (2, 4), (3, 5), (4, 6), (5, 6)\}$.

To calculate $w_{\mathbf{y}}(t)$, it suffices to store $\tilde{R}_i^{(j)}(t)$ and $\tilde{R}_{mn}^{(j,k)}(t)$ for all $i \in [N]$, $(m, n) \in \mathcal{E}$ and $j, k \in [M]$, which cost $(NM + |\mathcal{E}| M^2)$ storage.

Equation (3) and (4) require the knowledge of marginal probabilities $\mathbb{P}\{x_i^t = j\}$ and $\mathbb{P}\{x_m^t = j, x_n^t = k\}$. Next, we propose a polynomial time algorithm to calculate them. From (2), the marginal probability can be written as

$$\mathbb{P}\{x_i^t = j\} = (1 - \gamma) \frac{1}{W_t} \sum_{\mathbf{y}: y_i = j} w_{\mathbf{y}}(t) + \frac{\gamma}{M}.$$

Hence, without calculating W_t , we have

$$\mathbb{P}\{x_i^t = j\} - \frac{\gamma}{M} : \mathbb{P}\{x_i^t = k\} - \frac{\gamma}{M} = \sum_{\mathbf{y}: y_i = j} w_{\mathbf{y}}(t) : \sum_{\mathbf{y}: y_i = k} w_{\mathbf{y}}(t). \quad (8)$$

A. Tree-structure Task Graph

Now we focus on how to calculate the sum of weights in (8) efficiently. We start from tree-structure task graphs and solve the more general graphs by calling the proposed algorithm for trees a polynomial number of times.

We drop time index t in our derivation whenever the result holds for all time steps $t \in \{1, \dots, T\}$. For example, $\tilde{R}_i^{(j)} \equiv \tilde{R}_i^{(j)}(t)$. We assume that the task graph is a tree with N nodes where the N^{th} node is the root (final task). Let $e_i^{(j)} = \exp(\alpha \tilde{R}_i^{(j)})$ and $e_{mn}^{(jk)} = \exp(\alpha \tilde{R}_{mn}^{(jk)})$. Hence, the sum of exponents in (7) can be written as the product of $e_i^{(j)}$ and

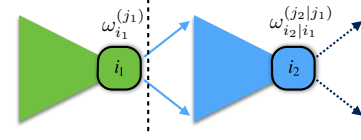


Fig. 3: A task graph consists of serial trees. To solve the sum of weights, $\omega_{i_2}^{(j_2)}$, we solve two trees rooted from i_1 and i_2 separately. When solving i_2 , we solve the conditional cases on all possible assignments of node i_1 .

$e_{mn}^{(jk)}$. That is,

$$\sum_{\mathbf{y}} w_{\mathbf{y}}(t) = \sum_{\mathbf{y}} \prod_{i=1}^N e_i^{(y_i)} \prod_{(m,n) \in \mathcal{E}} e_{mn}^{(y_m y_n)}.$$

For a node v , we use \mathcal{D}_v to denote the set of its descendants. Let the set \mathcal{E}_v denote the edges connecting its descendants. Formally,

$$\mathcal{E}_v = \{(m, n) \in \mathcal{E} | m \in \mathcal{D}_v, n \in \mathcal{D}_v \cup \{v\}\}.$$

The set of $|\mathcal{D}_v|$ -dimensional vectors, $\{y_m\}_{m \in \mathcal{D}_v}$, denotes all the possible assignments on its descendants. Finally, we define the sub-problem, $\omega_i^{(j)}$, which calculates the sum of weights of all possible assignment on task i 's descendants, given task i is assigned to device j . That is,

$$\omega_i^{(j)} = e_i^{(j)} \sum_{\{y_m\}_{m \in \mathcal{D}_i}} \prod_{m \in \mathcal{D}_i} e_m^{(y_m)} \prod_{(m,n) \in \mathcal{E}_i} e_{mn}^{(y_m y_n)}. \quad (9)$$

Figure 2 shows an example of a tree-structure task graph. Task 4 and 5 are the children of task 6. From (9), if we have $\omega_4^{(k)}$ and $\omega_5^{(l)}$ for all k and l , $\omega_6^{(j)}$ can be solved by

$$\omega_6^{(j)} = e_6^{(j)} \sum_{k,l} e_{46}^{(kj)} \omega_4^{(k)} e_{56}^{(lj)} \omega_5^{(l)}.$$

In general, the relation of weights between task i and its children $m \in \mathcal{N}_i$ is given by the following equation.

$$\begin{aligned} \omega_i^{(j)} &= e_i^{(j)} \sum_{\{y_m\}_{m \in \mathcal{N}_i}} \prod_{m \in \mathcal{N}_i} e_{mi}^{(y_m j)} \omega_m^{(y_m)} \\ &= e_i^{(j)} \prod_{m \in \mathcal{N}_i} \sum_{y_m \in [M]} e_{mi}^{(y_m j)} \omega_m^{(y_m)}. \end{aligned} \quad (10)$$

Algorithm 2 summarizes our approach to calculate the sum of weights of a tree-structure task graph. We first run breath first search (BFS) from the root node. Then we start solving the sub-problems from the last visited node such that when solving task i , it is guaranteed that all of its child tasks have been solved. Let d_{in} denote the maximum in-degree of G (i.e., the maximum number of in-coming edges of a node). Running BFS takes polynomial time. For each sub-problem, there are at most d_{in} products of summations over M terms. In total, Algorithm 2 solves NM sub-problems. Hence, Algorithm 2 runs in $\Theta(d_{in} NM^2)$ time.

B. More general task graphs

All of the nodes in a tree-structure task graph have only one out-going edge. For task graphs where there exists a node that has multiple out-going edges, we decompose the task graph into multiple trees and solve them separately and combine the solutions in the end. In the following, we use an example of a task graph that consists of serial trees to illustrate our approach.

Figure 3 shows a task graph that has two trees rooted by task i_1 and i_2 , respectively. Let the sub-problem, $\omega_{i_2|i_1}^{(j_2|j_1)}$, denote the sum of weights given that i_2 is assigned to j_2 and i_1 is assigned to j_1 . To find $\omega_{i_2|i_1}^{(j_2|j_1)}$, we follow Algorithm 2 but consider the assignment on task i_1 when solving the sub-problems on each leaf m . That is,

$$\omega_{(m|i_1)}^{j_m|j_1} = e_{i_1 m}^{(j_1 j_m)} e_m^{(j_m)}.$$

The sub-problem, $\omega_{i_2}^{(j_2)}$, now becomes the sum of weights of all possible assignment on task i_2 's descendants, including task 1's descendants, and is given by

$$\omega_{i_2}^{(j_2)} = \sum_{j_1 \in [M]} w_{i_2|i_1}^{(j_2|j_1)} w_{i_1}^{(j_1)}. \quad (11)$$

For a task graph that consists of serial trees rooted by i_1, \dots, i_n in order, we can solve $\omega_{i_r}^{(j_r)}$, given previously solved $\omega_{i_r|i_{r-1}}^{(j_r|j_{r-1})}$ and $\omega_{i_{r-1}}^{(j_{r-1})}$. From (11), to solve $\omega_{i_2}^{(j_2)}$, we have to solve $\omega_{i_2|i_1}^{(j_2|j_1)}$ for $j_1 \in \{1, \dots, M\}$. Hence, it takes $O(d_{in} n_1 M^2) + O(M d_{in} n_2 M^2)$ time, where n_1 (resp. n_2) is the number of nodes in tree i_1 (resp. i_2). Hence, to solve a serial-tree task graph, it takes $O(d_{in} N M^3)$ time.

Our approach can be generalized to more complicated DAGs, like the one that contains parallel chains of trees (parallel connection of Figure 3), in which we solve each chain independently and combine them from their common root N . Most of the real applications can be described by these families of DAGs where we have proposed polynomial time MABSTA to solve them. For example, in [15], the three benchmarks fall in the category of parallel chains of trees. In Wireless Sensor Networks, an application typically has a tree-structured workflow [16].

C. Marginal Probability

From (8), we can calculate the marginal probability $\mathbb{P}\{x_i^t = j\}$ if we can solve the sum of weights over all possible assignments given task i is assigned to device j . If task i is the root (node N), then Algorithm 2 solves $\omega_i^{(j)} = \sum_{\mathbf{y}: y_i = j} w_{\mathbf{y}}(t)$ exactly. If task i is not the root, we can still run Algorithm 2 to solve $[\omega_p^{(j')}]_{y_i = j}$, which fixes the assignment of task i to device j when solving from i 's parent p . That is,

$$[\omega_p^{(j')}]_{y_i = j} = e_p^{(j')} e_{ip}^{(j j')} \omega_i^{(j)} \prod_{m \in \mathcal{N}_p \setminus \{i\}} \sum_{y_m} e_{mp}^{(y_m j')} \omega_m^{(y_m)}.$$

Hence, in the end, we can solve $[\omega_N^{(j')}]_{y_i = j}$ from the root and

$$\sum_{\mathbf{y}: y_i = j} w_{\mathbf{y}}(t) = \sum_{j' \in [M]} [\omega_N^{(j')}]_{y_i = j}.$$

Similarly, the $\mathbb{P}\{x_m^t = j, x_n^t = k\}$ can be achieved by solving the conditional sub-problems on both tasks m and n .

Algorithm 3 Efficient Sampling Algorithm

```

1: procedure SAMPLING( $\gamma$ )
2:    $s \leftarrow \text{rand}()$   $\triangleright$  get a random number between 0 and 1
3:   if  $s < \gamma$  then
4:     pick an  $\mathbf{x} \in [M]^N$  uniformly
5:   else
6:     for  $i \leftarrow 1, \dots, N$  do
7:        $[\omega_i^{(j)}]_{x_1^t, \dots, x_{i-1}^t} \leftarrow \Omega(N, M, G)_{x_1^t, \dots, x_{i-1}^t}$ 
8:        $\mathbb{P}\{x_i^t = j | x_1^t, \dots, x_{i-1}^t\} \propto [\omega_i^{(j)}]_{x_1^t, \dots, x_{i-1}^t}$ 
9:     end for
10:  end if
11: end procedure

```

D. Sampling

As we can calculate the marginal probabilities efficiently, we propose an efficient sampling policy summarized in Algorithm 3. Algorithm 3 first selects a random number s between 0 and 1. If s is less than γ , it refers to the exploration phase, where MABSTA simply selects an arm uniformly. Otherwise, MABSTA selects an arm based on the probability distribution $p_{\mathbf{y}}(t)$, which can be written as

$$p_{\mathbf{y}}(t) = \mathbb{P}\{x_1^t = y_1\} \cdot \mathbb{P}\{x_2^t = y_2 | x_1^t = y_1\} \cdots \mathbb{P}\{x_N^t = y_N | x_1^t = y_1, \dots, x_{N-1}^t = y_{N-1}\}.$$

Hence, MABSTA assigns each task in order based on the conditional probability given the assignment on previous tasks. For each task i , the conditional probability can be calculate efficiently by running Algorithm 2 with fixed assignment on task $1, \dots, i-1$.

VI. NUMERICAL EVALUATION

In this section, we first examine how MABSTA adapts to dynamic environment. Then, we perform trace-data emulation to verify MABSTA's performance guarantee and compare it with other algorithms.

A. MABSTA's Adaptivity

Here we examine MABSTA's adaptivity to dynamic environment and compare it to the optimal strategy that relies on the existing profile. We use a two-device setup, where the task execution costs of the two devices are characterized by two different Markov processes. We neglect the channel communication cost so that the optimal strategy is the myopic strategy. That is, assigning the tasks to the device with the highest belief that it is in "good" state [17]. We run our experiment with an application that consists of 10 tasks and processes the incoming data frames one by one. The environment changes at the 100th frame, where the transition matrices of two Markov processes swap with each other. From Figure 4, there exists an optimal assignment (dashed line) so that the performance remains as good as it was before the 100th frame. However, myopic strategy, with the wrong information of the transition matrices, fails to adapt to the changes. From (2), MABSTA not only relies on the result of previous samples but also keeps exploring uniformly (with probability $\frac{\gamma}{M^N}$ for each arm). Hence, when the performance of one device degrades at 100th frame, the randomness enables MABSTA to explore another device and learn the changes.

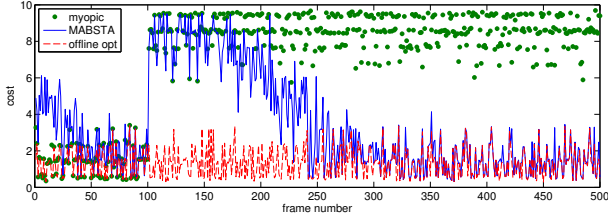


Fig. 4: MABSTA adapts to the changes at the 100th frame, while the myopic policy, relying on the old information of the environment, fails to adjust the task assignment.

TABLE I: Parameters Used in Trace-data measurement

Device ID	# of iterations	Device ID	# of iterations
18	$\mathcal{U}(14031, 32989)$	28	$\mathcal{U}(10839, 58526)$
21	$\mathcal{U}(37259, 54186)$	31	$\mathcal{U}(10868, 28770)$
22	$\mathcal{U}(23669, 65500)$	36	$\mathcal{U}(41467, 64191)$
24	$\mathcal{U}(61773, 65500)$	38	$\mathcal{U}(12386, 27992)$
26	$\mathcal{U}(19475, 44902)$	41	$\mathcal{U}(15447, 32423)$

B. Trace-data Emulation

To obtain trace data representative of a realistic environment, we run simulations on a large-scale wireless sensor network / IoT testbed. We create a network using 10 IEEE 802.15.4-based wireless embedded devices, and conduct a set of experiments to measure two performance characteristics utilized by MABSTA, namely channel conditions and computational resource availability. To assess the channel conditions, the time it takes to transfer 500 bytes of data between every pair of motes is measured. To assess the resource availability of each device, we measure the amount of time it takes to run a simulated task for a uniformly distributed number of iterations. The parameters of the distribution are shown in Table I. Since latency is positively correlated with device's energy consumption and the radio transmission power is kept constant in these experiments, it can also be used as an index

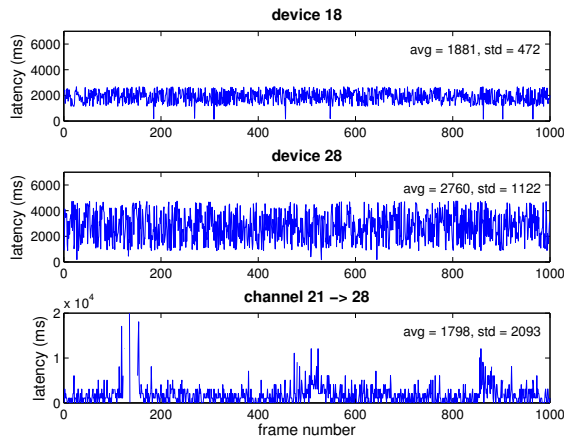


Fig. 5: Snapshots of measurement result: (a) device 18's computation latency (b) device 28's computation latency (c) transmission latency between them.

for energy cost. We use these samples as the reward sequences in the following emulation.

We present our evaluation as the regret compared to the offline optimal solution in (1). For real applications the regret can be extra energy consumption over all nodes, or extra processing latency over all data frames. Figure 6 validates MABSTA's performance guarantee for different problem sizes. From the cases we have considered, MABSTA's regret scales with $O(N^{1.5}M)$.

We further compare MABSTA with two other algorithms as shown in Figure 7 and Figure 8. Exp3 is proposed for adversarial MAB in [14]. Randomized baseline simply selects an arm uniformly for each data frame. Applying Exp3 to our task assignment problem results in the learning time grows exponentially with $O(M^N)$. Hence, Exp3 is not competitive in our scheme, in which the regret grows nearly linear with T as randomized baseline does. In addition to original MABSTA, we propose a more aggressive scheme by tuning γ provided in MABSTA. That is, for each frame t , setting

$$\gamma_t = \min \left\{ 1, \sqrt{\frac{M(N + |\mathcal{E}|M) \ln M^N}{(e-1)(N + |\mathcal{E}|)t}} \right\}. \quad (12)$$

From (2), the larger the γ , the more chance that MABSTA will do exploration. Hence, by exploring more aggressively at the beginning and exploiting the best arm as γ decreases with t , MABSTA with varying γ learns the environment even faster and remains competitive with the offline optimal solution, where the ratio reaches 0.9 at early stage. That is, after first 5000 frames, MABSTA already achieves the performance at least 90% of the optimal one. In sum, these empirical trace-based evaluations show that MABSTA scales well and outperforms the state of the art in adversarial online learning algorithms (EXP3). Moreover, it typically does significantly better in practice than the theoretical performance guarantee.

VII. APPLICATIONS TO WIRELESS DEVICE NETWORKS

MABSTA is widely applicable to many realistic scenarios, including in the following device networks.

A. Mobile Cloud Computing

Computational offloading - migrating intensive tasks to more resourceful servers, has been a widely-used approach to augment computing on a resource-constrained device [18]. The performance of computational offloading on cellular networks varies with channel and server dynamics. Instead of solving deterministic optimization based on profiling, like MAUI [19], or providing a heuristic without performance guarantee, like Odessa [15], MABSTA can be applied to learn the optimal offloading decision (task assignment) in dynamic environment.

B. Vehicular Ad Hoc Networks (VANETs)

Applications on VANETs are acquiring commercial relevance recently. These applications, like content downloading, rely on both vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications [20]. Computational offloading, or service discovery over VANETs are promising approaches with the help by road-side units and other vehicles [21]. How to leverage these intermittent connections and

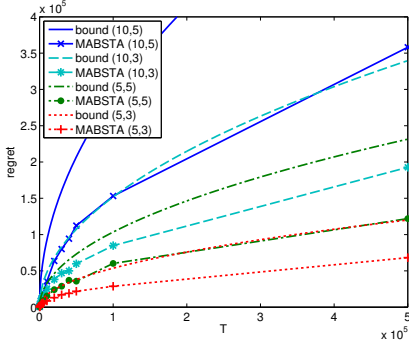


Fig. 6: MABSTA's performance with upper bounds provided by Corollary 1

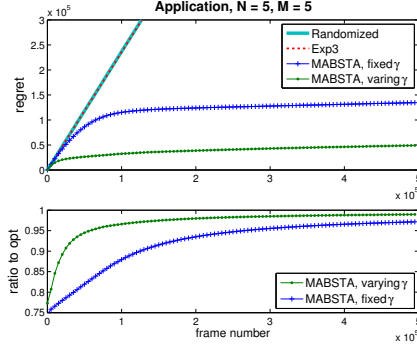


Fig. 7: MABSTA compared with other algorithms for 5-device network.

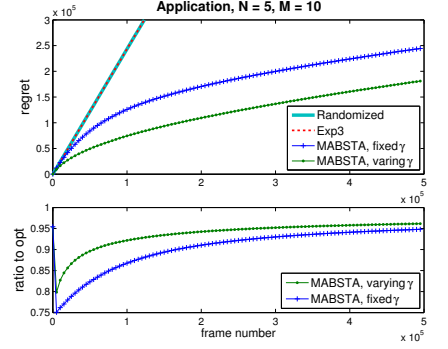


Fig. 8: MABSTA compared with other algorithms for 10-device network.

remote computational resources efficiently requires continuous run-time probing, which cannot be done by historical profiling due to fast-changing environment.

C. Wireless Sensor Networks and IoT

Wireless Sensor Networks (WSN) suffer from stringent energy usage on each node in real applications. These sensors are often equipped with functional microprocessors for some specific tasks. Hence, in some cases, WSN applications face the dilemma of pre-processing on less powerful devices or transmitting raw data to back-end processors [16]. Depending on channel conditions, MABSTA can adapt the strategies by assigning pre-processing tasks on front-end sensors when channel is bad, or simply forwarding raw data when channel is good. Moreover, MABSTA can also consider battery status so that the assignment strategy adapts to the battery remaining on each node in order to prolong network lifetime.

In the future IoT networks, fog computing is a concept similar to wireless distributed computing but scales to larger number of nodes and generalized heterogeneity on devices, communication protocols and deployment environment [22]. With available resources spread over the network, a high level programming model is necessary, where an interpreter takes care of task assignment and scheduling at run time [23]. No single stochastic process can model this highly heterogeneous scheme. As an approach to stochastic online learning optimization, MABSTA provides a scalable approach and performance guarantee for this highly dynamic run-time environment.

VIII. CONCLUSION

With increasing number of devices capable of computing and communicating, the concept of Wireless Distributed Computing enables complex applications which a single device cannot support individually. However, the intermittent and heterogeneous connections and diverse device behavior make the performance highly-variant with time. In this paper, we have proposed a new online learning formulation for wireless distributed computing that does not make any stationary stochastic assumptions about channels and devices. We have presented MABSTA, which, to the best of our knowledge, is the first online learning algorithm tailored to this class of problems. We have proved that MABSTA can be implemented efficiently and provides performance guarantee for all

dynamic environment. The trace-data emulation has shown that MABSTA is competitive to the optimal offline strategy and is adaptive to changes of the environment. Finally, we have identified several wireless distributed computing applications where MABSTA can be employed fruitfully.

APPENDIX A PROOF OF THEOREM 1

We first prove the following lemmas. We will use more condensed notations like $\hat{R}_i^{(y_i)}$ for $\hat{R}_i^{(y_i)}(t)$ and $\hat{R}_{mn}^{(y_m y_n)}$ for $\hat{R}_{mn}^{(y_m y_n)}(t)$ in the prove where the result holds for each t .

A. Proof of lemmas

Lemma 1.

$$\sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t) = \sum_{i=1}^N R_i^{(x_i^t)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(x_m^t x_n^t)}(t).$$

Proof:

$$\begin{aligned} \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t) &= \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}} \left(\sum_{i=1}^N \hat{R}_i^{(y_i)} + \sum_{(m,n) \in \mathcal{E}} \hat{R}_{mn}^{(y_m y_n)} \right) \\ &= \sum_i \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} + \sum_{(m,n)} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_{mn}^{(y_m y_n)}, \quad (14) \end{aligned}$$

where

$$\sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} = \sum_{\mathbf{y} \in \mathcal{C}_{ex}^i} p_{\mathbf{y}} \frac{R_i^{(x_i^t)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}} = R_i^{(x_i^t)},$$

and similarly,

$$\sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_{mn}^{(y_m y_n)} = R_{mn}^{(x_m^t x_n^t)}.$$

Applying the result to (14) completes the proof. \square

Lemma 2. For all $\mathbf{y} \in \mathcal{F}$, we have

$$\mathbb{E}\{\hat{R}_{\mathbf{y}}(t)\} = \sum_{i=1}^N R_i^{(y_i)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(y_m y_n)}(t).$$

Proof:

$$\begin{aligned} \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t)^2 &= \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}} \left(\sum_{i=1}^N \hat{R}_i^{(y_i)} + \sum_{(m,n) \in \mathcal{E}} \hat{R}_{mn}^{(y_m y_n)} \right)^2 \\ &= \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}} \left(\sum_{i,j} \hat{R}_i^{(y_i)} \hat{R}_j^{(y_j)} + \sum_{(m,n),(u,v)} \hat{R}_{mn}^{(y_m y_n)} \hat{R}_{uv}^{(y_u y_v)} + 2 \sum_i \sum_{(m,n)} \hat{R}_i^{(y_i)} \hat{R}_{mn}^{(y_m y_n)} \right) \end{aligned} \quad (13)$$

$$\mathbb{E}\{\hat{R}_{\mathbf{y}}(t)\} = \sum_{i=1}^N \mathbb{E}\{\hat{R}_i^{(y_i)}\} + \sum_{(m,n) \in \mathcal{E}} \mathbb{E}\{\hat{R}_{mn}^{(y_m y_n)}\}, \quad (15)$$

where

$$\mathbb{E}\{\hat{R}_i^{(y_i)}\} = \mathbb{P}\{x_i^t = y_i\} \frac{R_i^{(y_i)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}} = R_i^{(y_i)},$$

and similarly, $\mathbb{E}\{\hat{R}_{mn}^{(y_m y_n)}\} = R_{mn}^{(y_m y_n)}$. \square

Lemma 3. If $\mathcal{F} = \{\mathbf{x} \in [M]^N\}$, then for $M \geq 3$ and $|\mathcal{E}| \geq 3$,

$$\sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t)^2 \leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t).$$

Proof: We first expand the left-hand-side of the inequality as shown in (13) at the top of this page. In the following, we derive the upper bound for each term in (13) for all $i \in [N]$, $(m, n) \in \mathcal{E}$.

$$\begin{aligned} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} \hat{R}_j^{(y_j)} &= \sum_{\mathbf{y} \in \mathcal{C}_{ex}^i \cap \mathcal{C}_{ex}^j} p_{\mathbf{y}} \frac{R_i^{(x_i^t)} R_j^{(x_j^t)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}} \cdot \sum_{\mathbf{z} \in \mathcal{C}_{ex}^j} p_{\mathbf{z}}} \\ &\leq R_j^{(x_j^t)} \frac{R_i^{(x_i^t)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}} = R_j^{(x_j^t)} \hat{R}_i^{(x_i^t)} \leq \frac{1}{M^{N-1}} \sum_{\mathbf{y}} \hat{R}_i^{(y_i)} \end{aligned} \quad (16)$$

The first inequality in (16) follows by $\mathcal{C}_{ex}^i \cap \mathcal{C}_{ex}^j$ is a subset of \mathcal{C}_{ex}^j and the last inequality follows by $\hat{R}_i^{(y_i)} = \hat{R}_i^{(x_i^t)}$ for all \mathbf{y} in \mathcal{C}_{ex}^i . Hence,

$$\sum_{i,j} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} \hat{R}_j^{(y_j)} \leq \frac{1}{M^{N-2}} \sum_{\mathbf{y}} \sum_i \hat{R}_i^{(y_i)}. \quad (17)$$

Similarly,

$$\sum_{(m,n),(u,v)} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_{mn}^{(y_m y_n)} \hat{R}_{uv}^{(y_u y_v)} \leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{y}} \sum_{(m,n)} \hat{R}_{mn}^{(y_m y_n)}. \quad (18)$$

For the last term in (13), following the similar argument gives

$$\begin{aligned} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} \hat{R}_{mn}^{(y_m y_n)} &= \sum_{\mathbf{y} \in \mathcal{C}_{ex}^i \cap \mathcal{C}_{tx}^{mn}} p_{\mathbf{y}} \frac{R_i^{(x_i^t)} R_{mn}^{(x_m^t x_n^t)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}} \cdot \sum_{\mathbf{z} \in \mathcal{C}_{tx}^{mn}} p_{\mathbf{z}}} \\ &\leq R_{mn}^{(x_m^t x_n^t)} \frac{R_i^{(x_i^t)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}} = R_{mn}^{(x_m^t x_n^t)} \hat{R}_i^{(x_i^t)} \leq \frac{1}{M^{N-1}} \sum_{\mathbf{y}} \hat{R}_i^{(y_i)}. \end{aligned}$$

Hence,

$$\sum_i \sum_{(m,n)} \sum_{\mathbf{y}} p_{\mathbf{y}} \hat{R}_i^{(y_i)} \hat{R}_{mn}^{(y_m y_n)} \leq \frac{|\mathcal{E}|}{M^{N-1}} \sum_{\mathbf{y}} \sum_i \hat{R}_i^{(y_i)}. \quad (19)$$

Applying (17), (18) and (19) to (13) gives

$$\begin{aligned} \sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t)^2 &\leq \sum_{\mathbf{y} \in \mathcal{F}} \left[\sum_i \left(\frac{1}{M^{N-2}} + \frac{2|\mathcal{E}|}{M^{N-1}} \right) \hat{R}_i^{(y_i)} + \sum_{(m,n)} \frac{|\mathcal{E}|}{M^{N-2}} \hat{R}_{mn}^{(y_m y_n)} \right] \\ &\leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t). \end{aligned} \quad (20)$$

The last inequality follows by the fact that $\frac{1}{M^{N-2}} + \frac{2|\mathcal{E}|}{M^{N-1}} \leq \frac{|\mathcal{E}|}{M^{N-2}}$ for $M \geq 3$ and $|\mathcal{E}| \geq 3$. For $M = 2$, we have

$$\sum_{\mathbf{y} \in \mathcal{F}} p_{\mathbf{y}}(t) \hat{R}_{\mathbf{y}}(t)^2 \leq \frac{M + 2|\mathcal{E}|}{M^{N-1}} \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t).$$

Since we are interested in the regime where (20) holds, we will use this result in our proof of Theorem 1. \square

Lemma 4. Let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, if $\mathcal{F} = \{\mathbf{x} \in [M]^N\}$, then for all $\mathbf{y} \in \mathcal{F}$, all $t = 1, \dots, T$, we have $\alpha \hat{R}_{\mathbf{y}}(t) \leq 1$.

Proof: Since $|\mathcal{C}_{ex}^i| \geq M^{N-1}$ and $|\mathcal{C}_{tx}^{mn}| \geq M^{N-2}$ for all $i \in [N]$ and $(m, n) \in \mathcal{E}$, each term in $\hat{R}_{\mathbf{y}}(t)$ can be upper bounded as

$$\hat{R}_i^{(y_i)} \leq \frac{R_i^{(y_i)}}{\sum_{\mathbf{z} \in \mathcal{C}_{ex}^i} p_{\mathbf{z}}} \leq \frac{1}{M^{N-1} \frac{\gamma}{M^N}} = \frac{M}{\gamma}, \quad (21)$$

$$\hat{R}_i^{(y_{i-1} y_i)} \leq \frac{R_i^{(y_{i-1} y_i)}}{\sum_{\mathbf{z} \in \mathcal{C}_{tx}^i} p_{\mathbf{z}}} \leq \frac{1}{M^{N-2} \frac{\gamma}{M^N}} = \frac{M^2}{\gamma}. \quad (22)$$

Hence, we have

$$\begin{aligned} \hat{R}_{\mathbf{y}}(t) &= \sum_{i=1}^N \hat{R}_i^{(y_i)} + \sum_{(m,n) \in \mathcal{E}} \hat{R}_{mn}^{(y_m y_n)} \\ &\leq N \frac{M}{\gamma} + |\mathcal{E}| \frac{M^2}{\gamma} = \frac{M}{\gamma} (N + |\mathcal{E}| M). \end{aligned} \quad (23)$$

Let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, we achieve the result. \square

B. Proof of Theorem 1

Proof: Let $W_t = \sum_{\mathbf{y} \in \mathcal{F}} w_{\mathbf{y}}(t)$. We denote the sequence of decisions drawn at each frame as $\mathbf{x} = [\mathbf{x}^1, \dots, \mathbf{x}^T]$, where $\mathbf{x}^t \in \mathcal{F}$ denotes the arm drawn at step t . Then for all data frame t ,

$$\begin{aligned} \frac{W_{t+1}}{W_t} &= \sum_{\mathbf{y} \in \mathcal{F}} \frac{w_{\mathbf{y}}(t)}{W_t} \exp(\alpha \hat{R}_{(\mathbf{y})}(t)) \\ &= \sum_{\mathbf{y} \in \mathcal{F}} \frac{p_{\mathbf{y}}(t) - \frac{\gamma}{|\mathcal{F}|}}{1 - \gamma} \exp(\alpha \hat{R}_{(\mathbf{y})}(t)) \end{aligned}$$

$$\leq \sum_{\mathbf{y} \in \mathcal{F}} \frac{p_{\mathbf{y}}(t) - \frac{\gamma}{|\mathcal{F}|}}{1 - \gamma} \left(1 + \alpha \hat{R}_{(\mathbf{y})}(t) + (e - 2)\alpha^2 \hat{R}_{(\mathbf{y})}(t)^2 \right) \quad (24)$$

$$\leq 1 + \frac{\alpha}{1 - \gamma} \left(\sum_{i=1}^N R_i^{(x_i^t)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(x_m^t x_n^t)}(t) \right) + \frac{(e - 2)\alpha^2}{1 - \gamma} \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t). \quad (25)$$

Eq. (24) follows by the fact that $e^x \leq 1 + x + (e - 2)x^2$ for $x \leq 1$. Applying Lemma 1 and Lemma 3 we arrive at (25). Using $1 + x \leq e^x$ and taking logarithms at both sides,

$$\ln \frac{W_{t+1}}{W_t} \leq \frac{\alpha}{1 - \gamma} \left(\sum_{i=1}^N R_i^{(x_i^t)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(x_m^t x_n^t)}(t) \right) + \frac{(e - 2)\alpha^2}{1 - \gamma} \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t).$$

Taking summation from $t = 1$ to T gives

$$\ln \frac{W_{T+1}}{W_1} \leq \frac{\alpha}{1 - \gamma} \hat{R}_{total} + \frac{(e - 2)\alpha^2}{1 - \gamma} \frac{|\mathcal{E}|}{M^{N-2}} \sum_{t=1}^T \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t). \quad (26)$$

On the other hand,

$$\ln \frac{W_{T+1}}{W_1} \geq \ln \frac{w_{\mathbf{z}}(T+1)}{W_1} = \alpha \sum_{t=1}^T \hat{R}_{\mathbf{z}}(t) - \ln M^N, \quad \forall \mathbf{z} \in \mathcal{F}. \quad (27)$$

Combining (26) and (27) gives

$$\hat{R}_{total} \geq (1 - \gamma) \sum_{t=1}^T \hat{R}_{\mathbf{z}}(t) - (e - 2)\alpha \frac{|\mathcal{E}|}{M^{N-2}} \sum_{t=1}^T \sum_{\mathbf{y} \in \mathcal{F}} \hat{R}_{\mathbf{y}}(t) - \frac{\ln M^N}{\alpha}. \quad (28)$$

Eq. (28) holds for all $\mathbf{z} \in \mathcal{F}$. Choose \mathbf{x}^* to be the assignment strategy that maximizes the objective in (1). Now we take expectations on both sides based on $\mathbf{x}^1, \dots, \mathbf{x}^T$ and use Lemma 2. That is,

$$\sum_{t=1}^T \mathbb{E}\{\hat{R}_{\mathbf{x}^*}(t)\} = \sum_{t=1}^T \left(\sum_{i=1}^N R_i^{(x_i^*)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(x_m^* x_n^*)}(t) \right) = R_{total}^{max},$$

and

$$\begin{aligned} \sum_{t=1}^T \sum_{\mathbf{y} \in \mathcal{F}} \mathbb{E}\{\hat{R}_{\mathbf{y}}(t)\} &= \sum_{t=1}^T \sum_{\mathbf{y} \in \mathcal{F}} \left(\sum_{i=1}^N R_i^{(y_i)}(t) + \sum_{(m,n) \in \mathcal{E}} R_{mn}^{(y_m y_n)}(t) \right) \leq M^N R_{total}^{max}. \end{aligned}$$

Applying the result to (28) gives

$$\mathbb{E}\{\hat{R}_{total}\} \geq (1 - \gamma) R_{total}^{max} - |\mathcal{E}| M^2 (e - 2) \alpha R_{total}^{max} - \frac{\ln M^N}{\alpha}.$$

Let $\alpha = \frac{\gamma}{M(N + |\mathcal{E}|M)}$, we arrive at

$$R_{total}^{max} - \mathbb{E}\{\hat{R}_{total}\} \leq (e - 1)\gamma R_{total}^{max} + \frac{M(N + |\mathcal{E}|M) \ln M^N}{\gamma}.$$

□

REFERENCES

- [1] D. Datla, X. Chen, T. Tsou, S. Raghunandan, S. Shajedul Hasan, J. H. Reed, C. B. Dietrich, T. Bose, B. Fette, and J. Kim, "Wireless distributed computing: a survey of research challenges," *Communications Magazine, IEEE*, vol. 50, no. 1, pp. 144–152, 2012.
- [2] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Cwc: A distributed computing infrastructure using smartphones," *IEEE Transactions on Mobile Computing*, 2014.
- [3] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: computation offloading as a service for mobile devices," in *ACM MobiHoc*. ACM, 2014, pp. 287–296.
- [4] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *ACM SIGCOMM*. ACM, 2010, pp. 1–14.
- [5] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," in *ACM MobiHoc*. ACM, 2012, pp. 145–154.
- [6] X. Chen, S. Hasan, T. Bose, and J. H. Reed, "Cross-layer resource allocation for wireless distributed computing networks," in *RWS, IEEE*. IEEE, 2010, pp. 605–608.
- [7] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *IEEE INFOCOM*. IEEE, 2015.
- [8] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Foundations of Computer Science*. IEEE, 1995, pp. 322–331.
- [9] W. Dai, Y. Gai, and B. Krishnamachari, "Online learning for multi-channel opportunistic access over unknown markovian channels," in *IEEE SECON*. IEEE, 2014, pp. 64–71.
- [10] K. Liu and Q. Zhao, "Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5547–5567, 2010.
- [11] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *arXiv preprint arXiv:1204.5721*, 2012.
- [12] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
- [13] R. Ortner, D. Ryabko, P. Auer, and R. Munos, "Regret bounds for restless markov bandits," in *Algorithmic Learning Theory*. Springer, 2012, pp. 214–228.
- [14] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The non-stochastic multiarmed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.
- [15] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *ACM MobiSys*. ACM, 2011, pp. 43–56.
- [16] H. Viswanathan, E. K. Lee, and D. Pompili, "Enabling real-time in-situ processing of ubiquitous mobile-application workflows," in *IEEE MASS*. IEEE, 2013, pp. 324–332.
- [17] Y. M. Dirickx and L. P. Jennergren, "On the optimality of myopic policies in sequential decision problems," *Management Science*, vol. 21, no. 5, pp. 550–556, 1975.
- [18] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.
- [19] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *ACM MobiSys*. ACM, 2010, pp. 49–62.
- [20] M. Gerla and L. Kleinrock, "Vehicular networks and the future of the mobile internet," *Computer Networks*, vol. 55, no. 2, pp. 457–469, 2011.
- [21] B. Li, Y. Pei, H. Wu, Z. Liu, and H. Liu, "Computation offloading management for vehicular ad hoc cloud," in *Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 728–739.
- [22] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

- [23] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldchofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *ACM SIGCOMM workshop on Mobile cloud computing*. ACM, 2013, pp. 15–20.